Building an OAuth 2.0 Authorization Server with Flask and Authlib

# Introduction

OAuth 2.0 is an industry-standard protocol for authorization. It's widely used by many web services to grant limited access to users' resources. In this project, I built an OAuth 2.0 Authorization Server using Flask and Authlib. This server allows client applications to authenticate users and access their data using authorization codes and bearer tokens.

# Project Overview

The goal of this project was to set up an authorization server that issues authorization codes to clients and then exchanges those codes for access tokens. The access tokens are then used to grant limited access to protected resources.

# Key Components

## 1. Flask and Authlib Integration

The project uses Flask as the web framework and Authlib for handling OAuth 2.0 protocols. Authlib integrates seamlessly with Flask and provides built-in support for the OAuth 2.0 specification.

## 2. Client and User Models

The server includes models for clients and users. The Client class represents the application trying to access the resources, while the User class represents the end-user.

## 3. Authorization Code Grant

The server uses the authorization code grant flow, which involves redirecting the user to the authorization server to obtain an authorization code and then exchanging that code for an access token.

## 4. Bearer Token Validator

The server uses bearer tokens for authenticating API requests. The BearerTokenValidator ensures that incoming requests are authenticated correctly using the issued tokens.

# Implementation

## 1. Setting Up Flask and Authlib

Flask was used as the web framework, and Authlib was integrated to provide OAuth 2.0 support:

from flask import Flask, request, jsonify, redirect, session, render\_template\_string  
from authlib.integrations.flask\_oauth2 import AuthorizationServer, ResourceProtector

## 2. Client and User Classes

The Client and User classes were defined to model the client applications and users of the system:

class Client:  
 ...  
 # Attributes and methods for the client  
  
class User:  
 ...  
 # Attributes and methods for the user

## 3. Authorization Code Grant

The AuthorizationCodeGrant class was customized to handle the issuance and validation of authorization codes:

class MyAuthorizationCodeGrant(AuthorizationCodeGrant):  
 ...  
 # Custom implementation of the grant type

## 4. Bearer Token Validator

The MyBearerTokenValidator class was used to validate incoming bearer tokens for API requests:

class MyBearerTokenValidator(BearerTokenValidator):  
 ...  
 # Custom implementation of the token validator

## 5. Routes

The server has routes for handling authorization, token issuance, and protected resources:

@app.route('/authorize', methods=['GET', 'POST'])  
def authorize():  
 ...  
 # Authorization route  
  
@app.route('/token', methods=['POST'])  
def issue\_token():  
 ...  
 # Token issuance route  
  
@app.route('/api')  
@require\_oauth('profile')  
def api\_me():  
 ...  
 # Protected resource route

## 6. Running the Server

Finally, the server is set up to run and register the custom grant type:

if \_\_name\_\_ == '\_\_name\_\_':  
 server.register\_grant(MyAuthorizationCodeGrant)  
 app.run()

# Conclusion

This project demonstrates how to build an OAuth 2.0 authorization server using Flask and Authlib. The server handles the authorization code flow, issues bearer tokens, and protects resources using those tokens. This is a great starting point for building secure, scalable APIs that leverage OAuth 2.0 for authorization.

# Source Code

The complete source code for this project can be found [here](#).

# Author

This project was developed by [Your Name](#). I'm a software engineer with experience in backend development, security, and API design.